Sistema de Monitoreo Cardiaco en la Nube

Ing. Guillermo Castillo A., M.Sc. Miguel Yapur A.
Facultad de Ingeniería en Electricidad y Computación
Escuela Superior Politécnica del Litoral (ESPOL)
Campus Gustavo Galindo, Km 30.5 vía Perimetral
Apartado 09-01-5863. Guayaquil-Ecuador
gcastill@fiec.espol.edu.ec, myapur@fiec.espol.edu.ec

Resumen

La distribución de una señal eléctrica a través de la red, es posible utilizando técnicas y herramientas adecuadas. Este artículo muestra la capacidad de ciertos elementos de software y hardware, para la realización de un sistema de monitoreo cardiaco, que permite la distribución y visualización de una señal eléctrica, capturada por un electrocardiógrafo, a múltiples consumidores. Analizamos las técnicas utilizadas, así como los diferentes enfoques para la transmisión y manipulación de una señal a través de la red. Las señales generadas son procesadas por un servidor central, para luego ser distribuidas a múltiples clientes web utilizando WebSockets. De esta manera logramos que múltiples consumidores pudieran visualizar la actividad cardiaca capturada por un electrocardiógrafo, en tiempo real.

Palabras Claves: *sockets, WebSocket, python, electrocardiógrafo, tiempo real*

Abstract

The distribution of an electric signal is possible using the correct tools and techniques. This article shows how certain elements of software and hardware were used to help us build a heart rate monitor that is able to work in the cloud. This tools, allowed us to build a system capable of the distribution and visualization of an electrical signal, captured by an electrocardiograph. This signal was distributed, using the network and web technologies, to multiple consumers. We analyze the different techniques and approaches used for the transmission and manipulation of a signal through the network. The electrical signal is processed by a central server, which in turn distributes it among multiple clients using WebSockets. This allowed multiple consumers visualize the same signal, captured by an electrocardiograph, in real time.

Keywords*: sockets, WebSocket, python, electrocardiograph, real time*

1. Introducción

Un electrocardiógrafo es un dispositivo electrónico no invasivo, que permite leer la actividad eléctrica del corazón. Dicha actividad está representada por un voltaje a través del tiempo, y muchas veces es visualizado en un monitor electrónico, o impreso en papel para electrocardiogramas.

La introducción de dispositivos como el BeagleBone Black, que son micro-computadoras de bajo costo, con la capacidad de manejar entradas y salidas eléctricas, pone al alcance de los usuarios no electrónicos la capacidad de desarrollar herramientas que permiten manipular señales eléctricas en un ambiente de alto nivel.

Utilizando una de las entradas de voltaje analógico de éste dispositivo, fuimos capaces de tomar la señal emitida por un electrocardiógrafo, y enviarla a un servidor remoto a través de la red utilizando sockets.

El cliente EKG, es un pequeño programa escrito en python que es capaz de leer el voltaje del electrocardiógrafo y enviarlo a nuestro servidor periódicamente.

El servidor a su vez toma el voltaje y lo redistribuye a todos los clientes web conectados, utilizando el protocolo WebSocket.

El cliente web es el encargado de mostrar la señal y consiste en una aplicación construida utilizando tecnologías web. Este cliente se conecta a nuestro servidor a través del protocolo WebSocket y recibe la señal al tiempo que es capturada por el electrocardiógrafo.

2. Metodología

Utilizamos el lenguaje de programación *python* para crear una librería de distribución de mensajes de alto rendimiento a la que bautizamos *sephiroth.*

Esta librería utiliza sockets de red, y es capaz de crear dos puntos de comunicación, que no solo sirven para estandarizar el proceso de transmisión del mensaje, sino también para procesarlo y distribuirlo de manera dinámica.

En el cliente EKG utilizamos la librería Adafruit_BBIO para capturar el voltaje de una de las entradas analógicas en el BeagleBone Black. Éste voltaje es luego transmitido a través de la red, como una cadena de caracteres.

El servidor está compuesto por dos hilos que comparten una instancia de la clase *sephirot.endpoint*. Un proceso ejecuta el servidor, que recibe la señal capturada por el electrocardiógrafo, y la procesa. El segundo hilo consiste en un servidor de WebSockets que recibe peticiones por parte del cliente Web, y se encarga de registrar funciones anónimas en la instancia del servidor *sephiroth.endpoint*, el cual las ejecuta pasando como parámetros, entre otras cosas, el mensaje generado en el cliente EKG.

3. Sephiroth

Sephiroth es una librería escrita en python, que tiene como propósito establecer un canal de comunicación entre dos puntos, y facilitar la redistribución de la señal hacia potenciales consumidores.

El observador ingenuo, podría pensar que lo único necesario para establecer la comunicación, es crear dos sockets de red en cada extremo. Pero que sucede cuando queremos distribuir un mismo mensaje producido, a varios clientes en tiempo real?

Para lograr esto, es necesario establecer un mecanismo de redistribución. Una vez que el mensaje llega al servidor, éste debe ser capaz de replicarlo de manera dinámica a potenciales consumidores.

El servidor tiene la capacidad de registrar *handlers(*funciones que serán ejecutadas cada vez que el servidor recibe un mensaje), en tiempo de ejecución. Estas funciones son ejecutadas selectivamente por el servidor, dependiendo de la identificación única del cliente. Internamente, una instancia de un servidor, mantiene un registro de las funciones que han sido añadidas por cada cliente.

Las funciones forman parte de una lista, y son ejecutadas en el orden que fueron registradas. La ejecución ocurre en un mismo hilo, lo que puede ocasionar demoras en la redistribución, si es que una de las funciones tardara más de lo previsto. Futuras versiones, deberán manejar esto creando un *pool* de hilos que ejecuten cada función independientemente.

4. El cliente EKG

El cliente EKG es el encargado de digitalizar y enviar la señal a través de la red.

El cliente es un programa que utiliza la librería Adafruit_BBIO para tomar el valor del voltaje de una de las entradas analógicas de la tarjeta BeagleBone Black.

Cuando el cliente inicializa, éste entra en un ciclo infinito, intentando establecer una conexión con el servidor, cada tres segundos. Una vez establecida la conexión, el cliente envía una cadena de caracteres que representa su identificación única. Si ya existiese un cliente conectado con la misma identificación, el servidor rechaza y termina la conexión.

Terminado el proceso de inicialización, el programa entra en un nuevo ciclo infinito, esta vez para el envío de la señal a través de la red, a una frecuencia determinada. El mensaje enviado, consiste en una cadena de caracteres con la siguiente estructura:

Tabla 1. Estructura del mensaje

00000000004.21;0000.12	
00000000004.21	0000.12
Tiempo transcurrido	Voltaje

Si por algún motivo ocurriera una desconexión, el servidor regresaría al ciclo infinito de inicialización nuevamente.

5. El Servidor

El servidor consiste en un programa que inicia dos hilos, que comparten una instancia de la clase *sephiroth.endpoint*, en modo servidor.

El primer proceso se encarga de recibir conexiones de clientes EKG y procesar la señal recibida. También aquí, son ejecutadas las funciones lambda, registradas para cada cliente EKG.

El segundo proceso consiste en un servidor de WebSockets que recibe conexiones de clientes Web.

El servidor de WebSockets, espera una cadena de caracteres que consiste en la identificación única de un cliente EKG.

Con esta identificación, el servidor registra una función lambda en la instancia *sephiroth.endpoint*, y utilizando *closures* para acceder al método *send* de la instancia del servidor WebSocket, es capaz de transmitir la señal al cliente Web sin saberlo. La **Figura 1** muestra el pseudocódigo del proceso.

```
endpoint = sephiroth.endpoint
ws = WebSocket

ws.on_message(uid, msg)
    send_signal(msg):
        ws.send(msg)
    endpoint.add_handler(uid, send_signal)

endpoint.add_handler(uid, handler):
    endpoint.handlers[uid].add(handler)

endpoint.listen(uid, msg):
    for handler in endpoint.handlers[uid]:
        handler(msg)

p1 = Process(endpoint.listen).start()
p2 = Process(ws, endpoint).start()
```

Figura 1. Pseudocódigo del servidor

6. El cliente web

El cliente Web fue construido utilizando HTML5, CSS3 y Javascript. Utiliza la librería *smoothiecharts*, para la presentación de la señal.

La aplicación hace uso de la especificación del protocolo WebSocket, y está atada a la implementación de cada navegador web. La **Tabla 2** muestra la lista de navegadores y las versiones a partir de la cual soportan éste protocolo.

Tabla 1. Soporte de protocolo WebSockets

Navegador	Versión	Año
Chrome	4[3]	2010
Firefox	6[1]	2011
Safari	5[2]	2010
Opera	12.10[5]	2011
Android Browser	4.4[4]	2011

7. Conclusiones

La introducción de tarjetas como BeagleBone Black, permiten a muchos desarrolladores, no electrónicos, involucrarse en proyectos que requieren la manipulación de señales eléctricas. Podríamos concluir que gracias a este tipo de dispositivos, la brecha que separa el desarrollo de software y el desarrollo de soluciones que requieren automatización y manejo de señales eléctricas ha sido reducido.

Es posible transmitir y monitorear señales eléctricas a través de la red, en tiempo real, utilizando sockets en un lenguaje de alto nivel.

La introducción de la especificación del protocolo de WebSockets, permite la transmisión de datos entre un servidor y una aplicación web, a una alta frecuencia y en tiempo real.

8. Recomendaciones

Se recomienda el uso de la tarjeta BeagleBone Black, a usuarios que deseen manipular señales eléctricas, y que no necesariamente sean expertos en electrónica.

Recomendamos el uso de la librería *sephiroth*, para aplicaciones que necesiten establecer un canal de comunicación para la transmisión de datos, y que tengan la necesidad redistribuir o manipular los mensajes recibidos, de una manera dinámica y en tiempo real.

Recomendamos el uso de WebSockets, para toda aplicación que necesite transmitir y presentar datos, a múltiples usuarios, a altas frecuencias y en tiempo real.

9. Referencias

[1] "WebSockets (support in Firefox)". developer.mozilla.org. Mozilla Foundation. 2011-09-30. 2015-05-23.

[2] Katie Marsal (November 23, 2010). "Apple adds accelerometer, WebSockets support to Safari in iOS 4.2". AppleInsider.com. 2015-05-23.

[3] "Chromium bug 64470". code.google.com. Google. 2010-11-25. 2015-05-23.

[4] "Android 4.4 KikKat, the browser and the Chrome WebView". www.mobilexweb.com. 2015-05-23.

[5] "A hot Opera 12.50 summer-time snapshot". Opera Developer News. 2012-08-03. Archived from the original on 2012-08-05. 2015-05-23.